High Performance Radiation Transport Simulations: Preparing for TITAN

C. Baker, G. Davidson, T. M. Evans,
S. Hamilton, J. Jarrell and W. Joubert
Oak Ridge National Laboratory
Oak Ridge, TN USA

***Abstract*—In this paper we describe the Denovo code system. Denovo solves the six-dimensional, steady-state, linear Boltzmann transport equation, of central importance to nuclear technology applications such as reactor core analysis (neutronics), radiation shielding, nuclear forensics and radiation detection. The code features multiple spatial differencing schemes, state-of-the-art linear solvers, the Koch-Baker-Alcouffe (KBA) parallel-wavefront sweep algorithm for inverting the transport operator, a new multilevel energy decomposition method scaling to hundreds of thousands of processing cores, and a modern, novel code architecture that supports straightforward integration of new features. In this paper we discuss the performance of Denovo on the 20+ petaflop ORNL GPU-based system, Titan. We describe algorithms and techniques used to exploit the capabilities of Titan's heterogeneous compute node architecture and the challenges of obtaining good parallel performance for this sparse hyperbolic PDE solver containing inherently sequential computations. Numerical results demonstrating Denovo performance on early Titan hardware are presented.**

I. Introduction

Radiative transport modeling and simulation are central to many areas of nuclear technology, including reactor core analysis (neutronics), radiation shielding, nuclear forensics and radiation detection. Renewed interest in nuclear energy production and safety has increased the need for high fidelity nuclear simulations. Advanced simulation of nuclear reactors is needed to reduce capital and operator costs for reactors by enabling enhanced power uprates and lifetime extension, to reduce nuclear waste by enabling higher fuel burnups, and to enhance nuclear safety by enabling high-fidelity predictive capability for component and system performance. Expected growth in worldwide demand for energy in the coming years, as well as the impact of the Fukushima nuclear accident in Japan, highlights the need to increase the production of safe, abundant quantities of energy.

Accurate radiation transport calculations are extremely computationally intensive. Experience has shown that the vast majority of the computational expense of predictive nuclear energy simulations with coupled physics is due to the radiation transport solver applied to problems in six independent variables in quasi-static formulations. A transport solver that incorporates an adequately resolved discretization for all scales using current discrete models would require 10^{17}–10^{21} degrees of freedom per step, which is well beyond even exascale resources.

Denovo was developed to provide computational tools for the growing needs of nuclear modeling and simulation efforts [11]. The Denovo code embodies a fresh approach to solving the radiation transport problem, implementing modern best-of-breed solver algorithms in a rigorously engineered software code base. Denovo's solvers have been designed to scale from portable computing devices to multi-petaflop systems. Denovo is part of the SCALE [6] code system, has been used for high-profile science in multiple DOE INCITE [3] projects, has participated in the Joule code effort [14], and is the primary radiation transport code used in the multi-industry Consortium for Advanced Simulation of Light Water Reactors (CASL) [2] headquartered at Oak Ridge National Laboratory (ORNL).

Denovo was slated as one of six applications by the ORNL Center for Accelerated Application Readiness (CAAR) [23] for an early port to the 20+ petaflop ORNL Titan system. Titan is a hybrid GPU-based compute platform scheduled for completion in late 2012 as a staged upgrade of ORNL's Jaguar system. The port of Denovo to Titan required development of new algorithms to support increased parallelism and mapping of compute-intensive kernels to the GPU. The complexity of radiation transport algorithms makes it challenging to scale the methods to multi-petascale systems and beyond. Furthermore, obtaining high performance on these systems is challenging, since sparse hyperbolic PDE solvers are generally limited to very low computational intensities, and additionally, the algorithms induced by the physical problem are highly sequential and difficult to parallelize efficiently. Nonetheless, we have been able to achieve substantial performance efficiencies on petascale hardware with Denovo.

The remainder of this paper is organized as follows. A description of the radiation transport equations is given, followed by a description of the Denovo algorithms. The techniques for implementing Denovo on large heterogeneous systems are then described. Finally, numerical results are presented.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

II. Equations of Radiation Transport

The steady-state Boltzmann transport equation solved in Denovo for problems with multiplying (fissionable) media is

$$\begin{aligned}\hat{\mathbf{\Omega}} \cdot \nabla\psi(\mathbf{r},\mathbf{\Omega},E) + \sigma(\mathbf{r},E)\psi(\mathbf{r},\mathbf{\Omega},E) \\ = \int_0^\infty \int_{4\pi} \sigma_\mathrm{s}(\mathbf{r},\hat{\mathbf{\Omega}}'\cdot\hat{\mathbf{\Omega}},E'\to E)\psi(\mathbf{r},\mathbf{\Omega}',E')\,d\mathbf{\Omega}'dE' \\ + \frac{1}{k}\frac{\chi(E)}{4\pi}\int_0^\infty \int_{4\pi} \nu\sigma_\mathrm{f}(\mathbf{r},E')\psi(\mathbf{r},\mathbf{\Omega}',E')\,d\mathbf{\Omega}'dE' \,. \quad (1)\end{aligned}$$

Here, the state is defined by the angular flux ψ, and the independent variables are $\mathbf{r} = (x, y, z)$ in cm, $\mathbf{\Omega} = (\theta, \varphi)$ in str, and E in MeV representing space, angle, and energy respectively. This is an eigenvalue equation with k representing the multiplication of neutrons in successive fission generations. Equation (1) reverts to a regular integro-differential PDE in static media, in which case the last term is replaced by a source, $q(\mathbf{r},\mathbf{\Omega},E)$. This paper will deal exclusively with the eigenvalue form that is prevalent in nuclear reactor analysis.

Denovo solves this equation using the discrete ordinates (S_N) method. Full details are given in [11], but succinctly, the S_N method is a collocation method in angle. Energy is decomposed in groups, and the scattering integral is expanded in spherical harmonics. Finally, the spatial gradient operator can be discretized in several ways; Denovo supports both finite element and characteristic schemes for the spatial discretization. The multigroup S_N equations can be written in operator form as

$$\mathbf{L}\psi = \mathbf{MS}\phi + \frac{1}{k}\mathbf{M}\chi\mathbf{f}^T\phi \,. \quad (2)$$

The eigenvector is defined in angular flux moments, ϕ, that are related to the discrete angular flux through

$$\phi = \mathbf{D}\psi \,, \quad (3)$$

where $\mathbf{D}$ is the discrete-to-moment operator that integrates discrete angles into angular flux moments using quadrature rules. $\mathbf{L}$ is the first-order linear differential transport operator; $\mathbf{M}$ is the moment-to-discrete operator that projects angular flux moments into discrete angle space, and $\mathbf{S}$ is the group-to-group scattering matrix. $\mathbf{f}^T$ is the matrix of fission cross sections, and χ is the fission spectrum matrix. Operating by $\mathbf{DL}^{-1}$, defining $\mathbf{T} = \mathbf{DL}^{-1}$, and rearranging terms gives

$$(\mathbf{I} - \mathbf{TMS})\phi = \frac{1}{k}\mathbf{TMF}\phi \,, \quad (4)$$

where $\mathbf{F} = \chi\mathbf{f}^T$ is the fission matrix. The operator $\mathbf{L}^{-1}$ can be formed into a lower-triangular system if one sweeps the space-angle grid in the direction of neutron travel. The resulting transport *sweep* is the operation that is parallelized using the Koch-Baker-Alcouffe KBA sweep algorithm [8]. This matrix is never formed in practice; only the action of the operator on a vector, $y = \mathbf{L}^{-1}v$, is required. The fixed-source analog of Eq. (4) is

$$(\mathbf{I} - \mathbf{TMS})\phi = q \,. \quad (5)$$

This form of the equation is still important for eigenvalue problems because the inner matrix-vector products in many eigenvalue solvers reduce to solving multigroup, fixed-source equations.

III. Denovo Solvers and Algorithms

In most existing industry and laboratory research codes, Eq. (4) is solved using power iteration with non-linear, multigrid acceleration schemes. We start by defining an energy-independent eigenvector,

$$\Gamma = \mathbf{f}^T\phi \,. \quad (6)$$

Then, Eq. (4) can be written

$$\mathbf{A}\Gamma = k\Gamma \,, \quad (7)$$

with

$$\mathbf{A} = \mathbf{f}^T(\mathbf{I} - \mathbf{TMS})^{-1}\mathbf{TM}\chi \,. \quad (8)$$

Solving by power iteration proceeds as follows:

$$\Gamma^{\ell+1} = \frac{1}{k^\ell}\mathbf{A}\Gamma^\ell \,, \quad k^{\ell+1} = k^\ell\frac{\|\Gamma^{\ell+1}\|}{\|\Gamma^\ell\|} \,. \quad (9)$$

The operation $\mathbf{A}\Gamma^\ell$ necessarily involves solving multigroup problems with the same form as Eq. (5),

$$(\mathbf{I} - \mathbf{TMS})y^\ell = \mathbf{TM}\chi\Gamma^\ell \,. \quad (10)$$

The traditional method for solving Eq. (5) is Gauss-Seidel iteration, which is the same as solving G one-group space-angle problems per Gauss-Seidel iteration [17]. We refer to these one-group problems as *within-group* equations, and they have the following form:

$$(\mathbf{I} - \mathbf{TMS}_{gg})\phi_g = \bar{q}_g \,, \quad (11)$$

where $\bar{q}_g$ is an effective group source that includes all in-scattering and fission. When using Gauss-Seidel iteration over energy, the within-group solves represent a set of inner iterations.

Denovo provides several solvers (multiple Krylov methods and Richardson (source) iteration) for the within-group equations, and Diffusion Synthetic Acceleration (DSA) is available as a preconditioner for the Krylov options. When $\mathbf{S}$ is lower triangular (indicating downscattering only) the solution converges in one Gauss-Seidel iteration. However, when $\mathbf{S}$ is energy dense, the solution can converge very slowly. This structure occurs in the low-energy region of the spectrum where neutrons undergo Maxwellian upscattering.

Denovo provides a Gauss-Seidel solver for the multigroup equations that can be called within a power iteration or independently for fixed-source problems. While the implementation in Denovo is enhanced by applying Transport Two-Grid (TTG) acceleration to the Gauss-Seidel iterations and using GMRES(m) on the inner one-group solves [10], a fundamental limitation of these solvers is that they are parallelizable only over space-angle variables. The sequential nature of Gauss-Seidel prevents effective parallelization over energy. One could use parallel block-Jacobi iteration, but this option results in solvers that are too inefficient to be of practical use on most problems.

To remedy this, the Krylov solver framework in Denovo that is currently used in the inner one-group space-angle solves has been expanded to include energy. Including energy in the Krylov vectors enables the following benefits:

- The energy variable is decoupled allowing groups to be solved independently.
- Krylov subspace iteration is more efficient and robust than Gauss-Seidel iteration.
- Preconditioning a Krylov iteration is generally more robust and stable than accelerating Gauss-Seidel iterations.

Furthermore, including energy in the Krylov vector does not invalidate any of the existing sweep mechanics that are already implemented in Denovo, though it does make more parallelism potentially available to the sweep algorithm.

For multigroup fixed-source problems in the form of Eq. (5), application of a Krylov method requires the following two steps:

1) A full energy-space-angle sweep of the right-hand side source,
$$q = \mathbf{T}\hat{q}\,, \tag{12}$$
where $\hat{q}$ is an effective source that could be an external source (q_e), in the case of true fixed-source problems, or it could be a fission source iterate when nested inside power iteration.
2) A full energy-space-angle sweep each Krylov iteration to calculate the action of the operator on the latest iterate,
$$y^\ell = (\mathbf{I} - \mathbf{TMS})v^\ell\,, \tag{13}$$
where v^ℓ is the Krylov vector in iteration ℓ. We note that this vector is dimensioned $v^\ell \equiv \{v^\ell_{g,c,n,l,m}\}$ where g is the energy group, c is the cell index, n is the spatial unknown index in the cell, and (l,m) are the spherical harmonic moment indices.

For eigenvalue problems, we have implemented an Arnoldi Krylov subspace solver using the Trilinos [7] Anasazi package that can (1) take full advantage of the energy parallelism and (2) be more efficient than power iteration. Arnoldi iteration requires the eigenproblem to be written in standard form,
$$\mathbf{A}x = \lambda x\,. \tag{14}$$
Arnoldi iteration can implemented with either an energy-dependent eigenvector,
$$\mathbf{A}\phi = k\phi\,, \quad \mathbf{A} = (\mathbf{I} - \mathbf{TMS})^{-1}\mathbf{TMF}\,, \tag{15}$$
or energy-independent eigenvector
$$\mathbf{A}\Gamma = k\Gamma\,, \quad \mathbf{A} = \mathbf{f}^T(\mathbf{I} - \mathbf{TMS})^{-1}\mathbf{TM}\chi\,. \tag{16}$$
In either case, the implementation of Arnoldi iteration requires a matrix-vector multiply at each Krylov iteration of the form
$$y^\ell = \mathbf{A}v^\ell\,. \tag{17}$$
For the energy-dependent case, we have
$$z^\ell = \mathbf{TMF}v^\ell\,, \quad \text{(mat-vec and sweep)} \tag{18}$$
$$(\mathbf{I} - \mathbf{TMS})y^\ell = z^\ell\,. \quad \text{(fixed-source solve)} \tag{19}$$
Similarly, for the energy-independent eigenvector the steps are
$$z^\ell = \mathbf{TM}\chi v^\ell\,, \quad \text{(mat-vec multiply and sweep)} \tag{20}$$
$$(\mathbf{I} - \mathbf{TMS})y^\ell = z^\ell\,, \quad \text{(fixed-source solve)} \tag{21}$$
$$y^\ell \leftarrow \mathbf{f}^T y^\ell\,. \quad \text{(dot-product)} \tag{22}$$
Both methods require a fixed-source solve each iteration. We consider both the energy-dependent and independent approaches because we are uncertain a priori which method will be optimal for a given problem. The energy-dependent approach allows parallelization of the eigenvalue solve across energy at the expense of a much larger eigenvector. The energy-independent approach allows energy-domain parallelization over only the fixed-source solve, and the eigenvalue solve is parallel only over space-angle. However, this decomposition may be more efficient because the eigenvector is smaller, especially when work is dominated by the inner multigroup fixed-source solve.

IV. Achieving Parallel Scalability

A. The KBA Sweep Algorithm

The application of the operator $\mathbf{T} = \mathbf{DL}^{-1}$ to a vector is a highly sequential computation; for a regular (x, y, z) grid, the resulting value along a given angular direction depends on the results computed for the immediately previous values in the x, y and z directions. The computational pattern is identical to that of the well-known Gauss-Seidel and SOR linear solver methods applied to certain structured grid problems and is well-known to be difficult to parallelize [21].

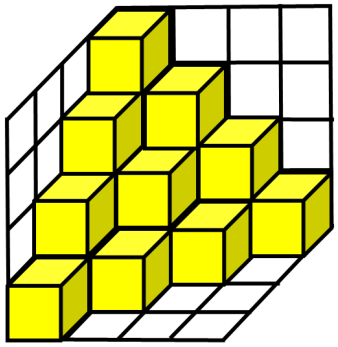


Fig. 1. KBA Algorithm Block-Wavefront.

An effective method for parallelizing this computation is the Koch-Baker-Alcouffe (KBA) algorithm [8]. For this algorithm, the 3D grid is decomposed into blocks, which are in turn arranged into block-wavefronts (see Fig. 1) numbered in sequence from the corner of the grid. Blocks are assigned to processors by assigning all blocks in the same vertical z stack to the same processor. Processors can then independently process each wavefront one at a time, with face communications after each wavefront is complete.

This technique effectively parallelizes the computation; however, parallel performance is stressed by two competing factors. The early and late parts of the computation perform at suboptimal efficiency, since these wavefronts have small

numbers of blocks, thus not all processors are active. The impact of this can be decreased by reducing the z-thickness of blocks, however this increases the total number of messages, increasing the costs of communication latency. The block z-thickness is a tuning parameter whose optimal size is problem- and hardware-specific.

B. Sweep-Based Parallelism

Denovo [11] uses the KBA wavefront algorithm to parallelize the space-angle transport sweeps shown in Eqs. (4) and (5). Unfortunately, KBA alone provides insufficient parallelism on very large systems, due to the parallel efficiency issue described above. The following analysis will demonstrate the scaling limitations of the KBA algorithm. The theoretical efficiency of KBA, ignoring machine latency, is [8]

$$\begin{aligned} \varepsilon_{\max} &= \frac{2MK}{2MK + K_b(I/I_b + J/J_b - 2)} \\ &= \frac{2MB_K}{2MB_K + P_I + P_J - 2}, \end{aligned} \tag{23}$$

where (I, J, K) are the number of cells in (x, y, z), respectively. The number of cells per domain in (x, y) is given by (I_b, J_b), and K_b is the number of cells per on-processor block in the z dimension. Then, P_I and P_J are the number of processors in the x and y directions, respectively; B_K is the number of blocks in the z direction on each domain; and M is the number of angles per octant.

We have done strong scaling studies on ORNL's Jaguar XT5 system with a $400 \times 400 \times 400$ cell mesh. The results are shown in Figs. 2 and 3.

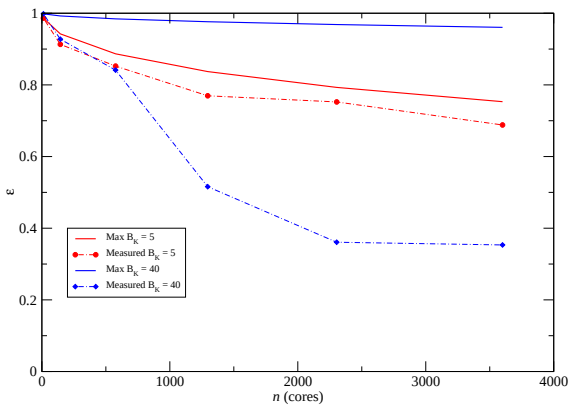


Fig. 2. Denovo strong scaling results on Jaguar XT5: strong scaling with $B_K = 40$ blocks (blue) and $B_K = 5$ blocks (red).

Figure 3 shows that while a very high ($> 90\%$) maximum theoretical parallel efficiency is estimated, this value is impossible to achieve in practice, due to hardware communication latencies. To get high theoretical efficiencies, the number of cells per block ($I_b \times J_b \times K_b$) becomes very small. This makes

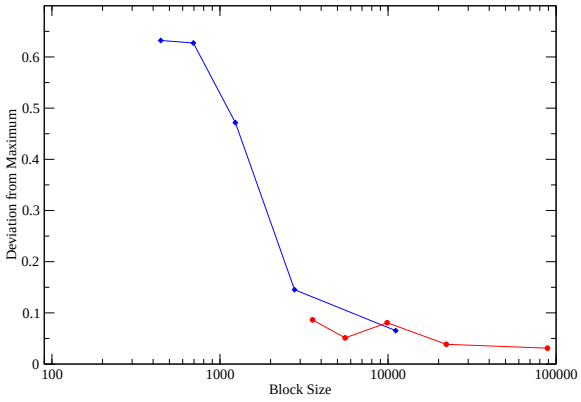


Fig. 3. Denovo strong scaling results on Jaguar XT5: the deviation from the theoretical maximum as a function of number of cells per block.

sense in the abstract because work is passed to subsequent blocks more rapidly. However, in reality the latency per message quickly overwhelms the theoretical prediction. When the number of cells per block becomes very small, data spend more time waiting in MPI message queues than working to solve the block. This effect is illustrated in Fig. 3. The deviation between the theoretical prediction and the measured efficiency becomes small as the block size grows. In summary, high theoretical efficiencies requiring small numbers of cells per block cannot be achieved, but theoretical efficiencies that result from larger block sizes can be realized.

The repercussions from this analysis are that the full extent of computational resources available on a petascale system such as Jaguar cannot be effectively utilized. The best efficiencies are obtained when the block size can be set greater than ~ 1500. Thus, for any given problem, the maximum number of cores is predetermined by the minimum block size. Even a 500 million cell problem will be limited to $15,000$–$20,000$ cores under these restrictions. To utilize the full resources of Titan, we must find additional variables to parallelize. Using the advanced solvers in Denovo, a multilevel decomposition over energy will provide the necessary parallelism.

C. Multilevel Parallel Decompositions

Having described the multigroup solvers in § III and discussed the limitations of parallelizing only the space-angle sweep, we now explain the parallel implementation of Denovo's energy-space-angle decomposition. The multilevel energy-space decomposition is illustrated in Fig. 4. In this decomposition, space is partitioned into *blocks*. Energy is partitioned into *sets*. Each set contains the full mesh (all of the blocks) such that KBA sweeps never cross set boundaries. Every (block, set) combination is termed a *domain*. The total number of domains is currently the same as the number of MPI [4] processes in a parallel job. The old Denovo space-

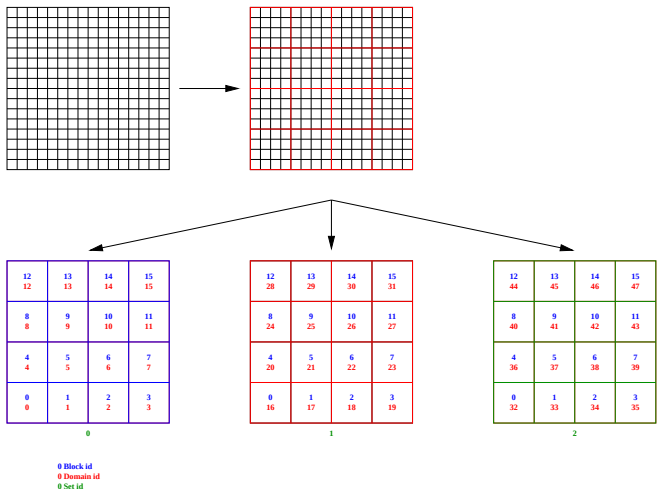


Fig. 4. Multilevel energy-space decomposition in Denovo. This example has 3 *sets*, each containing 16 *blocks*, resulting in 48 total *domains*. The block (blue), set (green), and domain (red) IDs are indicated by b, s, and d, respectively.

angle decomposition can be thought of as a single-set energy decomposition over $P_I \times P_J$ blocks. A benefit of the new energy-space partitioning is that all of the solvers described in § III can be implemented in the new decomposition using Denovo's existing space-angle sweep machinery.

To solve the multigroup equations, Eq. (13) implies a matrix-vector multiply of the form

$$s_g = \mathbf{S}_{g0} v_0 + \mathbf{S}_{g1} v_1 + \ldots + \mathbf{S}_{gG} v_G \,. \tag{24}$$

Instead of communicating all of the groups to each set so that the matrix-vector multiply can be completed locally, we replicate the source vector s. Then, each set performs its part of the matrix-vector multiply followed by a global reduction. The global reduction is performed using an MPI communicator that relates all blocks with the same index across sets. After the global reduction, each set has the complete source vector s, even though it will utilize only the components of s that are local to the set. As an alternative, a call to MPI_Reduce_scatter decreases the amount of communication by having each set receive only the part of s it needs. The cost of the reduction is typically a small part of the total runtime.

After calculating the sweep source s, the sweeps for each local group on a set can be performed without any interset communication. The sweeps require only communication between blocks within a set. The only exception to this rule is when the energy-independent version of Arnoldi is applied. In this case, the eigenvector must be summed across all sets in a manner analogous to that described above for calculating the sweep source.

V. Mapping to Heterogeneous Architectures

A. Algorithm

For typical Denovo runs, 80%–99% of execution time is spent in the KBA sweep algorithm. Thus, the parallel sweep is the obvious candidate for porting to the GPU. Generally the second highest consumer of runtime is the GMRES solver. GPU acceleration for GMRES is provided through the Trilinos [7] package, though this is limited by problem size due to the smaller GPU memory available for storing the required GMRES vectors.

Due to the specialized nature of the KBA algorithm, comparatively little work has been done previously to port this algorithm to GPUs. Furthermore, the work that has been done has mostly focused on the SWEEP3D benchmark [1]. The SWEEP3D work differs from the present work in that SWEEP3D is restricted to lower angular resolutions than are targeted by Denovo workflows and also is commonly run with many more gridcells per compute node than is practical when solving problems with higher angular resolution across thousands of compute nodes, the latter case being much more challenging for extracting large amounts of thread parallelism.

Gong et al. [12] mapped the SWEEP3D algorithm to NVIDIA GPUs by exploiting thread parallelism for spatial dimensions as well as the moment dimension for very large spatial grids. Lubeck et al. [19] parallelized the algorithm in space and vectorized in angle for the IBM Cell processor. Petrini et al. [22] also parallelized over a large spatial grid on a single IBM Cell processor and exploited parallelism in time, which is not available in the Denovo formulation. By contrast, the present work exploits accelerator parallelism over all available problem dimensions including space, octant, angle, moment and energy group, as will be shown below.

To obtain high performance for the sweep algorithm on NVIDIA GPUs it is necessary to (1) expose as much thread parallelism as possible, and (2) restructure the algorithm to maximize locality of reference, thus reducing the cost of data transfers. These are priorities not only for the current GPU porting effort but also for exascale preparedness, based on anticipated trends in computer hardware.

The Boltzmann equation offers rich opportunities for exploiting unrealized parallelism through its multiple problem dimensions. However, using this parallelism effectively is challenging due to the complex couplings between data elements in each dimension.

The sweep algorithm is composed of two levels. The outer level is a set of loops for computations across gridcells with associated dependencies as expressed in the KBA algorithm. The inner level is a set of computations within a gridcell. To promote data reuse, as much computation as possible must be permuted from outside the gridcell loops to occur at the gridcell level. The gridcell computation itself contains the following steps:

- A matrix-vector product corresponding to the matrix $\mathbf{M}$ of Eq. (4) is applied to transform the state vector from its native representation in terms of moments to an angles representation corresponding to physical flow directions.
- A small linear system is formed and solved with the state data.
- A matrix-vector product corresponding to the matrix $\mathbf{D}$ is performed to transform back from angles to moments.

This algorithm must be mapped to the Titan memory and parallelism hierarchy, which can be described as follows. Titan is a collection of Gemini network-connected nodes, each with an AMD 16-core Interlagos processor connected by PCIe-2 bus to an NVIDIA GPU processor. In the CUDA execution model [20], the GPU computations are arranged into a grid of fully independent threadblocks with access to GPU main memory. A threadblock is the largest unit of work for which a synchronization of threads is possible. Each threadblock also has access to its own section of "shared memory," essentially a programmable cache. Threadblocks are in turn divided into warps of 32 threads each which execute commands in lockstep. Threads each possess a localized subset of the register file.

The finer-grained levels of the parallelism hierarchy permit increasingly tight coupling of the constituent threads. Thus it is important to map each problem dimension to the level in the hierarchy appropriate to its required computations. The problem dimensions of the sweep algorithm are mapped to the GPU in the following fashion:

1) *Octant.* For KBA, each octant of angular directions is computed by an independent sweep process. This provides immediately a factor of eight in parallelism, with the octant dimension mapped to a CUDA grid axis. Since the results from the octants are eventually added together to the same state vector locations, some care must be taken to schedule the updates to avoid race conditions. For KBA steps for which the same block is updated by two octants, an overflow array is used to receive one of the results for later accumulation.
2) *Energy.* The Denovo multilevel energy formulation described above provides simultaneously all energy groups fully decoupled to the sweep algorithm, affording another axis of parallelism. This can be exploited as thread parallelism in the CUDA grid or threadblock dimensions, or as process parallelism across compute nodes in the form of energy sets described above, or both in combination.
3) *Space.* The same KBA approach to extracting spatial parallelism across nodes can be used for applying threads to spatial dimensions on the GPU. The on-GPU block is divided into subblocks arranged into subblock-hyperplanes sweeping from a corner of the block. Each block is assigned a compute thread. Since a synchronization is required after each subblock-hyperplane, in a manner analogous to the face communications of the macro KBA algorithm, the spatial axes must be mapped the the GPU in-threadblock.
4) *Moment and Angle.* Moment and angle axes are closely related via the matrix-vector products. Furthermore, neither axis is persistent; state information exists alternatively in moments form or angles form. Because of this, the axes are mapped to in-warp threads; because these threads operate in lockstep, the threads can be instantly redeployed alternatively between moments and angles. Registers and GPU shared memory are used for accumulating results of the matrix-vector products. Furthermore, since the entries of the matrices $\mathbf{M}$ and $\mathbf{D}$ are independent of most problem dimensions, the matrices are small enough to fit into GPU caches, resulting in BLAS-3-like performance.
5) *Gridcell Unknown.* The axis of unknowns corresponding to the spatial discretization values in a gridcell are tightly coupled and require different operations depending on the unknown. This axis is not easily adapted to thread parallelism and thus is left serial.

Figure 5 summarizes the parallelism scheme for each problem dimension.

B. Implementation

The sweep component of Denovo was fully rewritten to restructure the code for thread parallelism and heavy reuse of registers and caches. The NVIDIA CUDA runtime API was used to implement the GPU sweeper. All CUDA constructs are isolated in a set of C++ facade classes to make the code more portable to other accelerator APIs such as OpenCL [16]. Furthermore we expect that this code restructuring effort will make the future port to alternative parallel APIs such as OpenACC [5] or Intel Xeon Phi coprocessor offload directives [18] straightforward. A dual CPU/GPU programming model is used to enable generation of both CPU and GPU code versions from a single code base. Asynchronous PCIe-2 transfers and asynchronous MPI communications are used to maximize performance.

The sweep algorithm inner loops require a large number of registers for index calculations and floating point operations, whereas the amount of storage per thread for GPU registers, shared memory and caches is very limited. Optimizing the code under these constraints while maintaining good C++ programming style is challenging. The following techniques were applied to negotiate these issues:

- Careful use was made of inlining and templates to provide compile time optimization that avoids virtual function calls and to replace variables with constants.
- In some cases, indices were recomputed rather than of stored in order to minimize register usage.
- Memory for class member values was used sparingly to reduce redundancy between multiple objects of similar kind.
- Loop unrolling was used where appropriate, though this must be balanced against the resulting increase in register usage.

VI. Parallel Code Performance

A. System Configuration

Performance data are given for the following systems in anticipation of the final upgrade of Jaguar to Titan in late 2012:

- *Jaguar XT5*, a 2.3 petaflop Cray XT5 system in service through September 2011 consisting of 18,688 compute nodes, each with dual 2.6 GHz AMD 6-core Istanbul processors, DDR2-800 memory and Seastar 2+ interconnect.

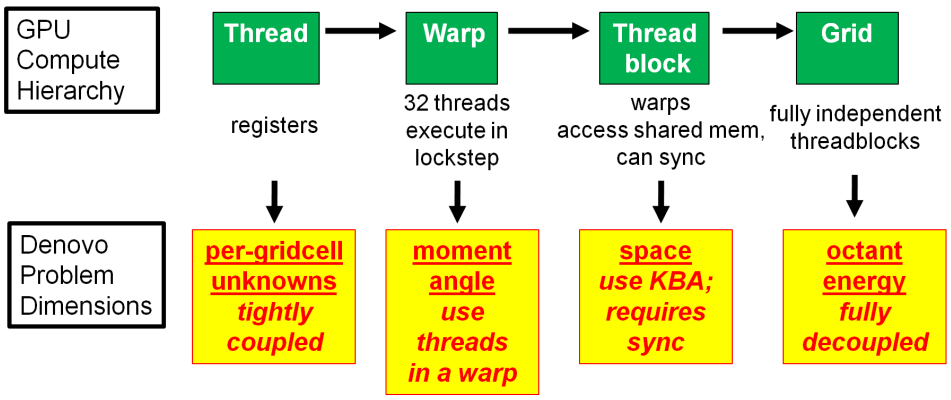


Fig. 5. GPU Parallelism Hierarchy.

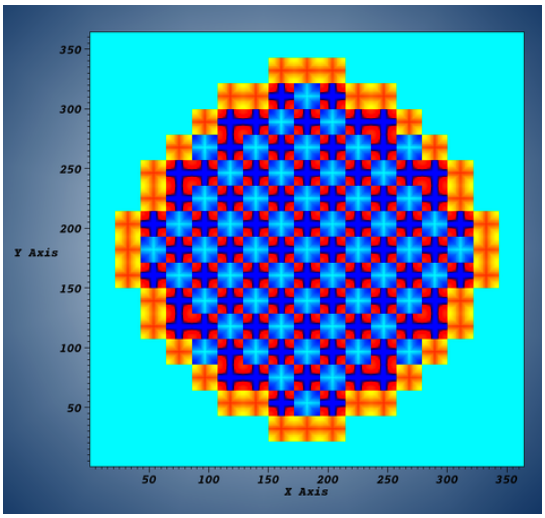
- *Jaguar XK6*, a 2.6 petaflop Cray XK6 system in service since February 2012 consisting of 18,688 compute nodes, each with a 2.2 GHz AMD 16-core Interlagos processor, DDR3-1600 memory and Gemini interconnect.
- *TitanDev*, a 10-cabinet partition of Jaguar XK6 with each of its 960 compute nodes populated with an NVIDIA Fermi X2090 GPU.

B. CPU-only Performance

To test the CPU-only performance of Denovo on the Jaguar XK6, we use the PWR-900 benchmark problem [9], which is a generic pressurized water reactor (PWR) model. This model was originally developed as a whole-core, pin-homogenized, two-energy group benchmark problem. In addition to this model, we have also altered the benchmark to make it a 44-group problem, which allows us to study the effects of multiple energy sets on the parallel scaling. Denovo is used to solve Eq. (4) in order to calculate the k-eigenvalue and the scalar flux throughout the core. The ability to solve pin-homogenized, whole-core problems with transport is the first step towards fully predictive reactor core modeling and simulation. To achieve first-principles predictive capability, each pin would be fully resolved in the whole-core 3-D model. This objective cannot be approached until we have demonstrated the ability to solve pin-homogenized 3-D reactor problems with full transport.

The model is a generalization of a Westinghouse PWR-900 core that has a core height of 4.2 m, an assembly height of 3.6 m, and a lattice pitch of 1.26 cm. The core features 289 assemblies, of which 157 are fuel and 132 are in the reflector. Each assembly contains a 17×17 array of homogenized fuel pins that are arranged with $1/4$ lattice symmetry. Three different fuel enrichments ranging from 1.5% to 3.25% are used in the assemblies. Each set of pin-homogenized cross sections contains 135 unique materials (45 pins at 3 levels of enrichment). The 2-D radial view of the core is shown in Fig. 6.

For Denovo, the model was discretized into $2 \times 2 \times 700$ spatial cells per homogenized fuel pin, resulting in a total mesh size of 233,858,800 cells. An angular quadrature



(a)

Fig. 6. 2-D radial cut of the reactor core. The low-enrichment assemblies are light blue, the medium enrichment assemblies are red/blue, and the high-enrichment assemblies are yellow/orange.

containing 128 angles, P_0 scattering (one angular moment), and step-characteristics spatial differencing was used for all calculations. These parameters yielded 29,933,926,400 DOF per energy group for each case.

Fig. 7 compares the different solvers for the two-group version of the PWR-900 benchmark. We use 17,424 spatial domains, which represents roughly the maximum number of spatial domains that can be used for this problem without encountering prohibitive network latency during a mesh sweep. To use more computing resource, the multilevel decomposition is required.

Two conclusions can be drawn from these results. First, the multigroup Krylov solver represents a significant improvement over the Gauss-Seidel solver, even though the Gauss-Seidel solver has Transport-Two Grid upscatter acceleration. Second, we see that the Arnoldi eigensolver, with either an energy-dependent or energy-independent eigenvector, has superior

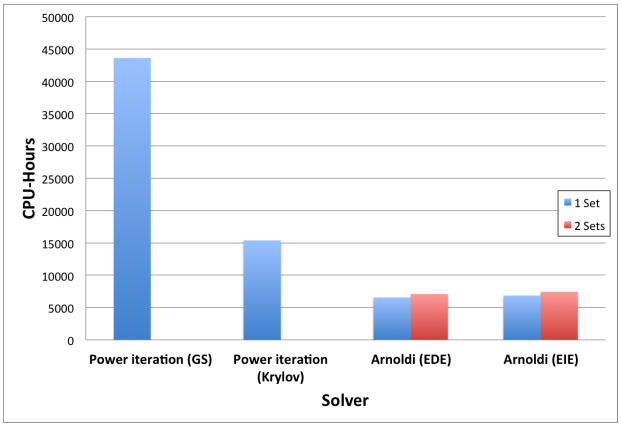


Fig. 7. Comparison of the four major eigenvalue solvers in Denovo: Power Iteration with Gauss-Seidel inner iterations and Transport-Two Grid acceleration, Power Iteration with Krylov inner iterations, Arnoldi with an energy-dependent eigenvector and Krylov inner iterations, and Arnoldi with an energy-independent eigenvector and Krylov inner iterations. 1- and 2-set comparisons demonstrate the efficiency of the energy set decomposition technique.

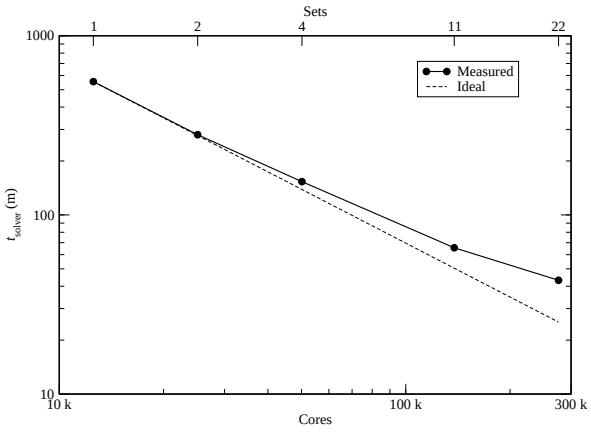


Fig. 8. Strong scaling results on a 44-group PWR-900 problem on the Jaguar XK6 machine. The number of sets is shown on the upper x-axis.

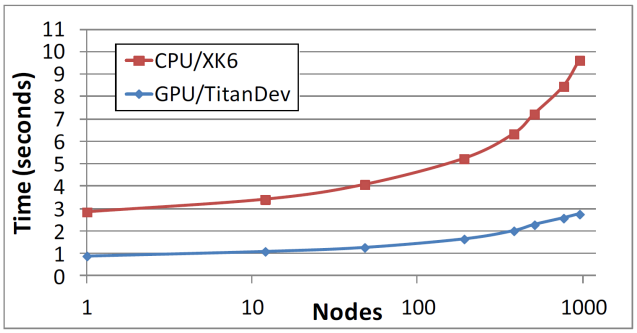


Fig. 9. Sweep Results on TitanDev for 16×32×64 gridcells per node, CPU vs. GPU performance.

performance over the power iteration methods. Furthermore, we see that using an energy-independent eigenvector does not add significantly to the computational expense. Therefore, we prefer the energy-independent eigenvector mode because it can significantly reduce the memory footprint for large problems.

To show the validity of this approach for fully resolved reactor problems, we must investigate the performance of the parallel algorithm and solvers on problems with more groups. We have run the same PWR-900 problem with 44 groups, resulting in 1,317,092,761,600 DOF, with 12,544 spatial domains and a varying number of energy sets, from one to 22. With 22 energy sets, the problem uses 275,968 cores, which is nearly the entire Jaguar XK6 machine. Fig. 8 shows the strong scaling behavior for this problem, with roughly 60% efficiency at 22 energy sets relative to one energy set.

We see that energy partitioning scales well even out to almost 300,000 cores. To make further gains in performace requires either that we implement more advanced algorithms that converge faster than the Krylov and Arnoldi solvers currently in use, or we make improvements to the cost of the sweep algorithm. It is the cost of the sweep that has motivated the GPU development work.

C. GPU Performance

Results are given here for the GPU sweep on TitanDev. For the first set of experiments, a single sweep operation is performed for a transport problem with 16 energy groups, one energy set, 16 moments and 256 angles using linear discontinuous elements with four unknowns per gridcell. Each compute node contains a spatial grid of 16×32×64 gridcells, and test cases are run on selections of $m \times n$ nodes in a weak scaling fashion.

For comparison, the same cases are also run with the CPU version of the code. As a side note, the CPU version of the GPU-enabled sweeper is roughly 2X faster than the original Denovo sweep code, a side benefit of performance improvement from the process of restructuring the code for better data reuse. The CUDA 4.0 toolkit and GCC 4.6.2 are used. All calculations are performed in double precision.

Results are shown in Fig. 9. An ideal weak scaling curve is flat; some expected degradation of ideal scaling exists as a result of the KBA algorithm's method of parallelizing the inherently sequential sweep operation as well as overhead from communication. At the largest node count, the GPU version is 3.5X faster than the CPU version, showing a significant performance gain compared to the CPU sweep version. By comparison, the GPU peak flop rate is 4.7X that of the CPU, and the peak memory bandwidth 3.5X, with same node interconnect speed in both cases, so a 3.5X code performance improvement is a reasonable expectation under these circumstances.

Figure 10 shows the FLOP rate measured on TitanDev. This case measures a sweep operation with 32 energy groups, 16 moments, 256 angles, linear discontinuous elements, and 16×32×256 spatial cells per compute node. The maximum

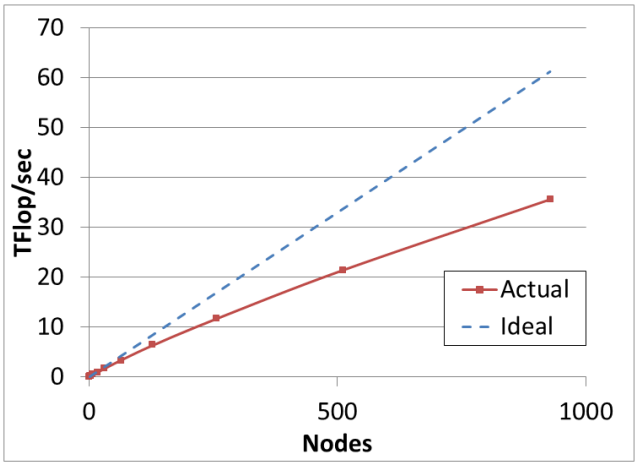


Fig. 10. GPU sweeper FLOP rate on TitanDev, 16×32×256 gridcells per node.

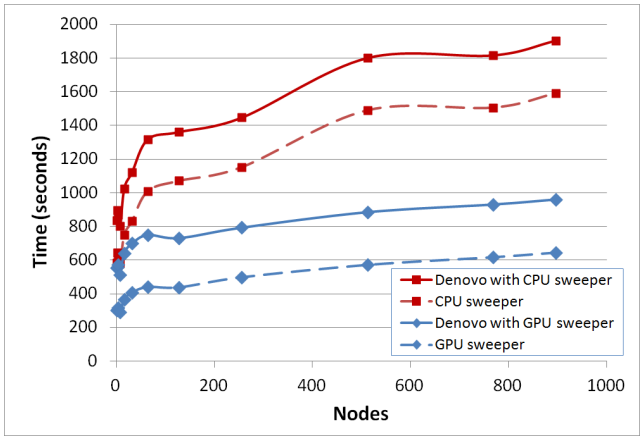


Fig. 11. Denovo reactor eigenvalue problem solve, runtime on TitanDev, weak scaling, with CPU vs. GPU sweeper.

performance is in excess of 35 TF/sec on 928 nodes of TitanDev. This is considered strong performance for a sparse PDE solver, with 58% parallel efficiency compared to single node performance. As described earlier, by use of energy set parallelism, it is not necessary to scale the sweep algorithm in space to the entire system; thus we anticipate similar performance per node on larger numbers of nodes, depending on the amount of energy set parallelism applied.

Finally, we present results for the Denovo code using the integrated GPU sweeper. For these experiments Denovo is used to solve a reactor eigenvalue problem in a weak scaling regime with 16 energy groups, one energy set, 16 moments, 256 angles, linear discontinuous elements, and 16×32×64 spatial cells per compute node. The tolerance for the Arnoldi outer iteration is 10^{-2}, while the tolerance for the inner GMRES solves is 10^{-4}. This algorithm regime is scalable, with the total GMRES iteration count increasing only 19% as the problem size is increased here by three orders of magnitude.

Results on TitanDev are shown in Fig. 11 for Denovo using either the CPU or GPU sweeper, with the time spent in the sweeper for each case also shown. These results use the CUDA 4.1 toolkit with GCC 4.6.2. The results show that the Denovo/GPU performance is not only substantially faster than the CPU-only case but also demonstrates excellent weak-scaling performance as the node count is increased.

VII. Conclusions

In this paper we have described the Denovo code system for solving the radiation transport problem. The primary algorithmic innovations described in this paper are: 1) a multilevel energy algorithm which permits efficient scaling of the KBA sweep solver to far greater numbers of compute nodes, and 2) a restructuring of the sweep algorithm at the node level to expose much higher levels of thread parallelism and memory locality which are needed for modern heterogeneous node architectures.

The need to solve the radiation transport problem accurately for nuclear energy simulations and other applications is becoming increasingly acute. The computational demands to solve these problems accurately requires computer hardware well beyond the scale of current systems. At the same time, the physics of these simulations make it difficult to solve these problems in parallel. With Denovo we have been able to implement algorithms to solve the radiation transport problem accurately and efficiently on modern heterogeneous high-end systems as well as prepare for the transition to exascale computing and beyond.

Acknowledgments

The authors would like to thank Matthew Bolitho, Paulius Micikevicius and John Roberts of NVIDIA and Jeff Larkin, James Schwarzmeier and Kevin Thomas of Cray Inc. for their assistance.

This research used resources of the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] —, "The ASCI SWEEP3D README File," http://wwwc3.lanl.gov/pal/software/sweep3d/sweep3d_readme.html.
[2] —, "The Consortium for Advanced Simulation of Light Water Reactors," http://www.casl.gov.
[3] —, "Department of Energy Leadership Computing: INCITE Program," http://www.doeleadershipcomputing.org.
[4] —, "The Message Passing Interface (MPI) standard," http://www.mcs.anl.gov/research/projects/mpi.
[5] —, "OpenACC: Directives for Accelerators," http://www.openacc-standard.org.
[6] —, "SCALE: A Modular Code System for Performing Standardized Computer Analyses for Licensing Evaluations," ORNL/TM-2005/39, Oak Ridge National Laboratory, 2009.
[7] —, "The Trilinos Project," http://trilinos.sandia.gov.
[8] Baker, R. S. and Koch, K. R. "An S_n Algorithm for the Massively Parallel CM-200 Computer," *Nuclear Science and Engineering*, **128**, 312–320, 1998.
[9] Courau, T. "Specifications of a 3D PWR Core Benchmark for Radiation Transport," Technical Report CR-128/2009/014 EDF-SA, Électricté de France, 2009.

[10] Evans, T. M., Clarno, K. T., and Morel, J. E. "A Transport Acceleration Scheme for Multigroup Discrete Ordinates with Upscattering." *Nuclear Science and Engineering*, **165**, 292–304, 2010.

[11] Evans, T. M., Stafford, A. S., Slaybaugh, R. N., and Clarno, K. T. "Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE." *Nuclear Technology*, **171**, 171–200, 2010.

[12] Gong, C., Liu, J., Gong, Z., Qin, J., and Xie, J., "Optimizing Sweep3D for Graphic Processor Unit," in *Algorithms and Architectures for Parallel Processing,* Lecture Notes in Computer Science, Vol. 6081, pp. 416–426, Springer, 2010.

[13] Joubert, W. "Developing a 3-D Sweep Radiation Transport Code for Large-scale GPU Systems," presented at *SIAM Conference on Computational Science and Engineering,* Reno, NV, Feb 28 – Mar 4, 2011.

[14] Kothe, D., and Roche, K., "The ASCR Joule Metric for Computational Effectiveness: An Update on Recent Activities and Outcomes," March 2010, http://science.energy.gov/sitecore/shell/Controls/˜/media/ascr/ ascac/pdf/meetings/mar10/Roche_kothe.pdf.

[15] Kerbyson, D. J., Lang, M. and Pakin, S., "Adapting wave-front algorithms to efficiently utilize systems with deep communication hierarchies," *Parallel Computing,* vol. 37, September 2011, pp. 550–561.

[16] Khronos Group, "OpenCL: The open standard for parallel programming of heterogeneous systems," http://www.khronos.org/opencl.

[17] Lewis, E. E. and Miller, W. F. "Computational Methods of Neutron Transport," American Nuclear Society, Inc., LaGrange Park, IL, 1993.

[18] Li, S. and Jin, J., "A Unified Approach to Heterogeneous Programming with Intel Multicore Processors and the Intel Many Integrated Core Architecture," Intel Developer Forum 2011, http://software.intel.com/sites/billboard/sites/default/files/BJ11_SFTS009_102_ENG_v02.pdf.

[19] Lubeck, O., Lang, M., Srinivasan, R., and Johnson, G., "Implementation and performance modeling of deterministic particle transport (Sweep3D) on the IBM Cell/B.E.," *Journal of Scientific Programming* **17**(1-2), 199-208, 2009.

[20] NVIDIA Corporation, "Parallel Programming and Computing Platform: CUDA," http://www.nvidia.com/object/cuda_home_new.html.

[21] Ortega, J. M. and Voigt, R. G., *Solution of Partial Differential Equations on Vector and Parallel Computers*. Philadelphia: SIAM, p. 41, 1987.

[22] Petrini, F., Fossum, G., Fernandez, J., Verbanescu, A. L., Kistler, M., and Perrone, M., "Multicore Surprises: Lessons Learned from Optimizing Sweep3D on the Cell Broadband Engine," *Proceedings of IPDPS*, 2007, pp. 1–10.

[23] Turner, J., "ORNL Center for Accelerated Application Readiness (CAAR)," First Hybrid Multicore Consortium Workshop, San Francisco, CA, 20–22 Jan. 2010, http://computing.ornl.gov/HMC/documents/HMC_ORNL_JT.pdf.